VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Optimalizace cest ve skladu

Optimal routes in a storehouse

2012 Adam Silber

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Zadání diplomové práce

Student: **Bc. Adam Silber**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 1103T031 Výpočetní matematika

Téma: Optimalizace cest ve skladu
Optimal routes in a storehouse

Zásady pro vypracování:

Smyslem diplomové práce je řešení reálného problému optimalizace cest ve skladu medicínského materiálu. Ve skladu jsou pravoúhlé uličky a mezi nimi ostrůvky zvané biny. Všechny uličky jsou průjedné po celé délce. Pro každou zakázku se použijí komponenty z několika binů a je potřeba navrhnout nejkratší trať pro jeden vysokozdvižný vozík tak, aby během jedné jízdy po skladu posbíral cca 10 až 60 komponent (krabic) a přivezl je na vyhrazené místo pro kompletaci.
Každá komponenta se vyskytuje na jediném místě ve skladu a mapa skladu (rozmístění jednotlivých komponent) se může v čase měnit podle toho, kam navezou nové krabice.
Cílem práce je navržení a případně i realizace algoritmu pro navržení optimální trasy (případně více tras, pokud se jedná o objemnější zakázku) vysokozvdižných vozíků po skladu za splnění předpokladů specifikovaných v zadání.

Seznam doporučené odborné literatury:

* K.H.Rosen: Discrete Mathematics and Its Applications, New York NY, (2007), ISBN-10 0-07-288008-2.
* D.L.Applegate, R.E.Bixby, V. Chvátal & W.J.Cook: The Traveling Salesman Problem: A Computational Study, Princeton University Press, 2006, ISBN-10: 0691129932.
* D.B.West: Introduction to graph theory, Upper Saddle River NJ, (2001), ISBN 0-13-014400-2.
* Dále odborné články dle pokynů vedoucího.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Petr Kovář, Ph.D.**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

doc. RNDr. Jiří Bouchala, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.*

Tato diplomová práce obsahuje citlivé informace firmy Mölnlycke Health Care v Karviné a z tohoto důvodu bude rozdělena na veřejnou a neveřejnou část. Veškeré citlivé informace budou zařazeny do neveřejné části a ta nebude uvedena v odevzdané diplomové práci, ani v její elektronické podobě. Tato část bude k dispozici pouze firmě Mölnlycke Health Care a zkoušející komisi při obhajobě diplomové práce. Autor sám si je rovněž tohoto faktu vědom a bez svolení Mölnlycke Health Care nebude obsah neveřejné části práce nikde šířit.

V Ostravě 4. května 2012 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 4. května 2012 .....

Velice děkuji vedoucímu mé diplomové práce Mgr. Petru Kovářovi, Ph.D. za jeho rady, návrhy a připomínky, které mi výrazně pomohly a za čas, který mi v průběhu psaní práce věnoval. Také děkuji panu Janu Korponaiovi, který se nám věnoval při naší návštěvě skladu, a s kterým jsme problém konzultovali prostřednictvím emailu. Děkuji také mým rodičům, kteří mě během studia trpělivě podporovali.

Abstrakt

Tato diplomová práce se zabývá řešením reálného problému, optimalizovat cestu vysokozdvižného vozíku ve skladu, problém byl zadaný firmou Mölnlycke Health Care za účelem zefektivnění kompletace objednávek. Zadaný problém lze přeformulovat jako problém obchodního cestujícího a v práci je následně řešen pomocí aproximačních algoritmů. Konkrétní řešení potom vychází z Christofidova algoritmu. Práce demonstruje možnou spolupráci akademické obce se soukromým sektorem.

Klíčová slova: optimalizace skladu, problém obchodního cestujícího, Christofidův algoritmus

Abstract

This thesis deals with the problem of finding optimal routes in a Mölnlycke Health Care's storehouse. The problem is similar to the traveling salesman problem and it is solved using approximation algorithms. For the solution we modiffy the Christofides algorithm. This thesis also shows possibility of cooperation between university and industry.

Keywords: warehouse optimization, storehouse, travelling salesman problem, Christofides algorithm

Seznam použitých zkratek a symbolů

$G(V, E)$	–	Graf G s množinou vrcholů V a množinou hran E
$V(G)$	–	Množina vrcholů gragu G
$E(G)$	–	Množina hran gragu G
TSP	–	Problém obchodního cestujícího

Obsah

Seznam obrázků

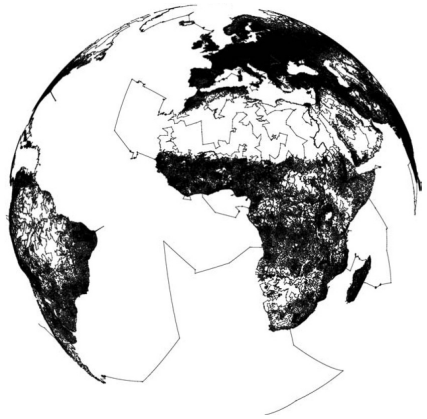
1 Úvod

Optimalizační problém, problém nalezení nejlepšího ze všech přípustných řešení, je často formulován jako hledání minima ceny, použitého materiálu nebo energie. V této diplomové práci se budeme zabývat hledáním minimální, nejkratší trasy, která musí obsahovat určité povinné zastávky.

Zadání diplomové práce vzniklo na žádost firmy Mölnlycke Health Care Klinipro, resp. jejího zaměstnance Jana Korponaie. Mölnlycke Health Care je přední světový dodavatel jednorázových chirurgických prostředků, výrobků pro hojení ran a služeb pro zdravotnictví. Firma vyrábí a skladuje zdravotnický materiál. Během kompletace objednávek pro různé zákazníky, je nutné jednotlivé komponenty z objednávky posbírat a svézt ze skladu na určité místo.

Optimalizace bude obsahovat popis algoritmu, který jednotlivé položky z objednávky sestaví v takovém pořadí, že obsluha skladu urazí během jejich vyzvednutí co nejkratší cestu a nebude si zbytečně zacházet, v našem případě zajíždět s vysokozdvižným vozíkem. Tím myslíme, že položky budou seřazeny postupně v závislosti na pořadí, ve kterém se nacházejí při průchodu nejkratší trasy, případně jedné z nejkratších tras. Zdali touto cestou obsluha vysokozdvižného vozíku skutečně pojede či ne už algoritmus zajistit nedovede. Rovněž se neočekává implementace algoritmu, který v případě budoucí implementace musí komunikovat se stávajícím obslužným systémem skladu.

Při přípravě objednávky je ve skladu potřeba navštívit určité regály ”biny”s požadovanou komponentou a se všemi vyzvednutými komponentami se potom vrátit na místo kompletace. Takto zadaná úloha, má blízko ke známému problému obchodního cestujícího - TSP (travelling salesman problem), který uvedeme v kapitole 4. Jedná se o diskrétní optimalizační problém nalezení nejkratší možné cesty procházející všemi zadanými body na mapě, každým právě jednou.



Obrázek 1: World tour z knihy The Traveling Salesman Problem [2]

Tento problém je již poměrně důkladně prostudován a existují i algoritmy které ho řeší. Zásadní potíž však je, že problém obchodního cestujícího patří mezi tzv. NP-těžké úlohy, pro které se nepředpokládá existence polynomiálního algoritmu a v obecném případě není známo jak nalézt přesné řešení v rozumném čase. Tuto komplikaci bude třeba v práci zahrnout do úvahy, jelikož není možné, aby se na nalezení optimálního pořadí čekalo několik hodin, ve skutečnosti by měl být výsledek k dispozici v řádu vteřin, nebo lépe zlomků vteřiny.

Při řešení zadaného problému budeme diskutovat známé algoritmy a používat obraty a pojmy z teorie Grafů, z tohoto důvodu je do práce zahrnuta kapitola 3 Definice a pojmy z teorie grafů, která slouží k objasnění používaných pojmů. V kapitole 2 si představíme zadaný problém důkladněji a seznámíme se zde s terminologií obsluhy skladu, kterou potom budeme dále používat, abychom se vyhnuli případným nejasnostem. Podíváme se na známe algoritmy a v kapitole 5 zhodnotíme, který a za jakých případných úprav by mohl být pro náš problém ten nejvhodnější. Samotný algoritmus potom popíšeme v kapitole 6.

2 Zformulování problému

Diplomová práce řeší reálný problém optimalizace cesty při kompletaci objednávky ve skladu medicínského materiálu firmy Mölnlycke Health Care v Karviné. Zadání problému bylo specifikováno při osobní návštěvě skladu, které se kromě autora diplomové práce zúčastnil také vedoucí diplomové práce Petr Kovář. Po celou dobu naší návštěvy se nám věnoval zaměstnanec firmy Mölnlycke Health Care, Jan Korponai, který nás společně se svým kolegou skladem provedli a zodpověděli naše otázky.

Skladové prostory a rozmístění regálů uvnitř je patrné z obrázku **??** na straně **??**, uvedeném v neveřejné kapitole 9. Tyto informace jsou považovány firmou Mölnlycke Health Care za citlivé a nejsou proto uvedeny ve veřejné části práce. Sklad je tvořen sítí pravoúhlých uliček oddělenými regály, ve kterých jsou uloženy krabice s medicínským materiálem. Ve skladu se kompletují zakázky obsahující různý počet komponent, které je třeba před kompletací svézt na jedno místo pomocí vysokozdvižného vozíku. Cestu tohoto vysokozdvižného vozíku budeme chtít optimalizovat tak, aby se minimalizovala vzdálenost, kterou vysokozdvižný vozík při nabírání materiálu pro objednávku ujede, a tím pádem snížila doba nutná pro přípravu, svoz komponent.

2.1 Současný stav

Současný stav je následující. Ve skladu firmy Mölnlycke Health Care v Karviné operuje několik (jejich přesný počet není pro řešení diplomové práce důležitý) vysokozdvižných vozíků. Komponenty každé objednávky se vytisknou na tzv. "picking list", který se předá obsluze vysokozdvižného vozíku. Ta podle svého vlastního uvážení zvolí trasu, kterou sklad projede a nabere všechny požadované komponenty. Vlastní sklad je systematicky rozdělen na čtyři části: sklad A, sklad B, sklad C a sklad D. Ve skladu A se nacházejí komponenty, které se využívají nejčastěji a rovněž se v něm dokují připravené objednávky.

Picking list je seznam krabic, obsahující jednotlivé komponenty, které jsou potřebné pro danou objednávku. Obsahuje informace o velikosti krabice, expirační době materiálu a pozici binu, kde je krabice umístěna ve skladu.

Bin je místo pro uložení krabice s komponentami ve skladu. Je jednoznačně určeno pomocí kódu, který je součástí "picking listu", označující sklad, uličku, patro a pozici v uličce.

Umístění komponenty ve skladu je dáno jednoznačně, jelikož se při kompletaci objednávky trvá na tom, že se použijí komponenty s nejkratší expirační dobou, aby ty jejihž trvanlivost je delší mohly být použity pro pozdější objednávku.

Po osobní konzultaci autora diplomové práce a vedoucího diplomové práce se zadavatelem problému ze strany Mölnlycke Health Care při návštěně skladu v Karviné, souhlasily všechny zůčastněné strany se zjednodušením, které nebere v potaz možnost, kdy je objednávka příliš rozsáhlá na to, aby mohla být svezena ze skladu při jediné jízdě vysokozdvižného vozíku. Tato eventualita se podle zaměstnanců skladu vyskytuje velice vyjímečně.

2.2 Následující postup

V této diplomové práci se budeme snažit navrhnout algoritmus, který najde pro každou objednávku v rozumném čase optimální cestu pro vysokozvižný vozík mezi jednotlivými biny obsahující komponenty z objednávky ve skladu. Tento algoritmus potom jednotlivé biny podle této cesty seřadí a vytiskne na "picking list", který obsluze vysokozdvižného vozíku ukáže v jakém pořadí je nejvhodnější komponenty sbírat. Cesta mezi jednotlivými komponentami už v "picking listu"zachycena není, ale vzhledem k pravoúhlé síti uliček a zkušenosti zaměstnanců se předpokládá využití jedné z najkratších cest.

Nalezení nejkratší možné cesty procházející všemi zadanými body ve skladu resp. nalezení nejkratší hamiltonovské kružnice na podgrafu je NP- úplný problém a předpokládá[1] se, že žádný "rychlý", (s polynomiální složitostí) deterministický algoritmus, neexistuje. Nalezení nejkratší možné cesty procházející všemi zadanými body ve skladu je úloha, která je velice blízká problému obchodního cestujícího. Problém obchodního cestujícího, který si podrobněji představíme v kapitole 4, by se dal formulovat jako problém nalezení nejkratší možné cesty, která projde zadanými místy, každým právě jednou. Tento problém je tedy skutečně skoro stejný s tím, který máme my a budeme tedy vycházet se známých algoritmů pro řešení problému obchodního cestujícího. Nemusíme být přitom tak přísní na podmínku, aby jsme každé místo navštívili právě jednou, ale můžeme se spokojit s podmínkou, aby každé místo (které bude zadáno) bylo navštíveno alespoň jednou.

Aby skutečně došlo k zefektivnění výroby ve skladu není možné, aby algoritmus hledal optimální řešení příliš dlouho. Bude proto třeba najít vhodný algoritmus, jiný než porovnávání délky všech možných cest ve skladu se složitostí $n!$. Naskýtá se nám zde několik možností:

a) Užití nedeterministického algoritmu. Nedeterministické nebo heuristické algoritmy davají většinou přibližný výsledek a při opakovaném stejném vstupu můžou dávat rozdílné výsledky, díky možnosti volby kroku. Neprocházejí všechny možnosti a za cenu určité nejistoty jsou výpočetně méně náročné.

[1] Jedná se o jeden z otevřených problémů tisíciletí (anglicky Millenium Prize Problems), za vyřešení každého z nich vypsal Clay Mathematics Institute odměnu jednoho milionu dolarů.

Těmto algoritmům se ale budeme snažit vyhnout, jelikož by bylo vhodné, aby pro dvě stejné objednávky dal algoritmus stejné výsledky.

b) Zjednodušení. Graf popisující strukturu skladu se můžeme pokusit zjednodušit, zanedbat faktory, které nehrajou výraznou roli, využít vhodné závislosti, které se ve skladu vyskytují. Při zjednodušení může nastat zkreslení reálné situace, toto může být výhodné pokud za cenu vnesení drobné nepřesnosti získáme významnou úsporu výpočetní náročnosti problému. Touto cestou se budeme snažit jít.

c) Využití symetrie grafu. Pokud nechceme použít nedeterministický algoritmus ale deterministický, který může být pro obecné grafy výpočetně příliš náročný, můžeme využít skutečnosti, že graf na kterém budeme problém řešit, dobře známe. Nemusíme procházet všecny možné cesty, protože některé nebudou vůbec existovat. Rovněž můžeme využít opakující se struktury v grafu.

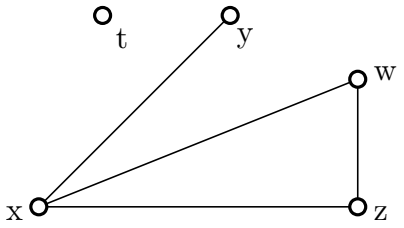
3 Definice a pojmy z teorie grafů

V této kapitole zavedeme pojmy, věty a definice, na které se budeme v dalším textu odkazovat. Abychom nemuseli text práce neustále přerušovat definicemi vysvětlující právě používané pojmy, je většina z nich uvedena v této kapitole. Pojmy jsou řazeny tak, aby k pochopení právě uvedené definice stačila znalost předchozích. Pojmy a definice jsou z oblasti teorie grafů, nejedná se však v žádném případě o úplný výčet všech definic této matematické disciplíny, což by snad ani nebylo možné, ale pouze o přehled těch definic, které budeme v práci využívat, nebo jejichž znalost by se nám mohla hodit. Některé věty a definice byly převzaty ze skript teorie grafů [1].

V značení pojmů se snažím držet běžně používané symboliky, známé ze skript či odborných článků. Někdy je ale, vzhledem k potřebě rozlišovat stejné pojmy u různých grafů (například počet vrcholů nebo pravidelnost grafu) použito nestandardní značení. Zkratky a značení většinou vycházejí z anglické terminologie, např. označení množiny vrcholů V (*vertices*) a hran E (*edges*).

Definice 3.1 *Jednoduchý graf G je uspořádaná dvojice (V, E), kde V je neprázdná množina* vrcholů *a E je nějaká podmnožina dvouprvkových podmnožin množiny V. Prvkům E říkáme* hrany.

Graf je nejčastěji zadán diagramem, kde se vrcholy znázorňují jako body a hrany jako křivky spojující tyto body.



Obrázek 2: Příklad jednoduchého grafu G s množinou vrcholů $V = \{t, w, x, y, z\}$ a množinou hran $E = \{\{x, y\}, \{x, w\}, \{x, z\}, \{w, z\}\}$.

Definice 3.2 *Množinu všech vrcholů grafu G budeme značit $V(G)$ a množinu všech hran $E(G)$.*

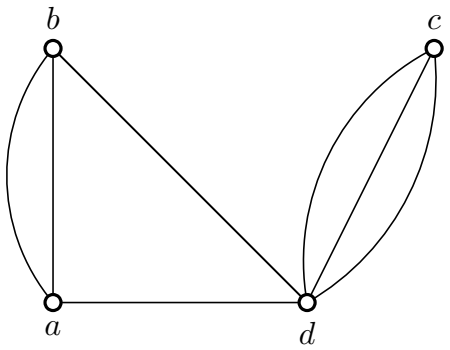
V textu někdy pracujeme s libovolným grafem G, na jehož parametry nemáme žádné požadavky. Chceme-li u grafu specifikovat počet vrcholů $n = |V(G)|$, řekneme, že jde o graf na n vrcholech nebo o graf řádu n.

Definice 3.3 *Dva vrcholy $x, y \in V(G)$ jsou* sousední, *když existuje hrana $xy \in E(G)$.*

Na obrázku 2 jsou například vrcholy x a z sousední, zatímco vrcholy w a y sousední nejsou. Sousedním vrcholům se také říká závislé a nesousedním nezávislé vrcholy.

Definice 3.4 *Řekneme, že hrana $e = \{x, y\} \in E(G)$ je* incidentní *s vrcholem x právě tehdy, když $x \in e$. Vrcholy x, y nazveme koncové vrcholy hrany e.*

Definice 3.5 *Graf, ve kterém mohou být dva různé vrcholy spojeny více než jednou hranou, nazveme* multigraf. *Hranám které mají stejné oba koncové vrcholy se říká násobné hrany, multihrany nebo paralelní hrany.*



Obrázek 3: Příklad multigrafu s množinou vrcholů $V = \{a, b, c, d\}$.

Definice 3.6 *Řekneme, že graf $H = (V', E')$ je* podgrafem *grafu $G = (V, E)$, jestliže $V \subseteq V'$ a současně $E \subseteq E'$.*

Jestliže o grafu H řekneme, že je podgrafem grafu G, potom také graf G můžeme označit za nadgraf grafu H.

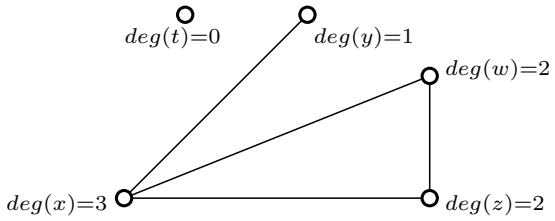
Definice 3.7 *Podgraf $F = (V', E')$ grafu $G = (V, E)$, se nazývá* faktor *grafu G jestliže $V = V'$.*

Definice 3.8 *Doplněk grafu $G = (V, E)$ je graf $\overline{G} = (V, F)$, kde $F = \binom{V}{2} \setminus E$ a $V(G) = V(\overline{G})$. $\binom{V}{2}$ značí všechny dvouprvkové podmnožiny množiny $V(G)$.*

Doplněk grafu G tedy obsahuje všechny vrcholy grafu G, a každý vrchol $x \in V(\overline{G})$ je sousední s právě těmi vrcholy, se kterými v grafu G sousední není.

Definice 3.9 Stupeň vrcholu x *je roven počtu všech hran, které jsou s vrcholem x incidentní. Stupeň vrcholu x značíme $deg(x)$.*

Stupeň vrcholu x současně dává informaci s kolika vrcholy je vrchol x sousední. Je tedy zřejmé, že stupeň žádného vrcholu nemůže být větší než počet vrcholů v grafu bez vrcholu samotného, jelikož sám se sebou nemůže být sousední. Nejvyšší možný stupeň, kterého může vrcholu $x \in V(G)$ nabývat, je $deg(x) = |V(G)| - 1$. Toto platí pouze pro grafy, nikoliv už však pro multigrafy.



Obrázek 4: Stupně vrcholů grafu G

Věta 3.1 (Princip sudosti) *Mějme libovolný graf G s vrcholy $v_1, v_2, \ldots, v_n$, kde $n \geq 1$. Počet hran grafu G označíme $h(G) = |E(G)|$. Potom platí*

$$\sum_{v_i \in V(G)} deg(v_i) = 2\, h(G).$$

Důkaz věty 3.1 je možné najít v každých skriptech teorie grafů. Všimneme-li si, že každá hrana zvyšuje stupeň vrcholu o 1 právě u dvou vrcholů, zjistíme, že v každém grafu je součet stupňů vrcholů všech vrcholů grafu roven dvojnásobku počtu hran.

Poznámka 3.1 Věta 3.1 nám říká, že v grafu nemůže být lichý počet vrcholů lichého stupně (počet hran musí být celé číslo).

Definice 3.10 *[1] Graf nazveme* sudý, *jestliže má všechny vrcholy sudého stupně. Podobně budeme mluvit o sudých multigrafech.*

Definice 3.11 *[1]* Sled *v grafu je taková posloupnost vrcholů a hran $(v_0, e_1, v_1, e_2, v_2 \ldots, e_n, v_n)$, že hrana e_i má koncové vrcholy v_{i-1} a v_i pro všechna $i = 1, 2, \ldots, n$. Délku sledu definujeme jako počet hran obsažených ve sledu.*

Definice 3.12 *[1]* Tah *je sled ve kterém se žádná hrana neopakuje. Tah s počátečním vrcholem u a koncovým vrcholem v budeme nazývat (u, v)-tah. Délku tahu definujeme jako počet hran obsažených v (u, v)-tahu.*

Definice 3.13 *[1]* Cesta *je sled ve kterém se neopakuje žádný vrchol. Cestu s počátečním vrcholem u a koncovým vrcholem v budeme nazývat (u, v)-cesta. Délku cesty definujeme jako počet hran obsažených v (u, v)-cestě.*

Definice 3.14 *Graf se nazývá* souvislý, *jestliže mezi každými dvěma jeho vrcholy x, y existuje (x, y)-cesta.*

Definice 3.15 *[1] Uzavřený tah v grafu G, který obsahuje všechny hrany a všechny vrcholy grafu G, se nazývá (*uzavřený*)* eulerovský tah. *Graf, ve kterém existuje uzavřený eulerovský tah, se nazývá* eulerovský graf *a říkáme, že graf G lze nakreslit jedním tahem.*

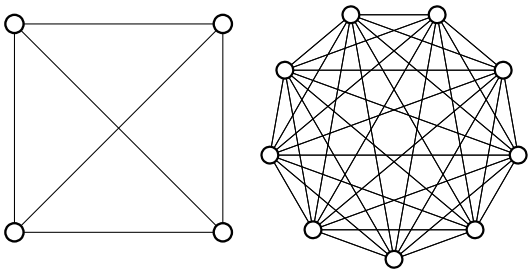
Následující věta dává nutnou a postačující podmínku existence eulerovského tahu v daném grafu.

Věta 3.2 (Eulerovský tah) *Graf G je* eulerovský *právě tehdy, když je* sudý *a* souvislý.

Poznámka 3.2 (Eulerovský tah) Věta 3.2 platí analogicky i pro multigrafy. Multigraf G je *eulerovský* právě tehdy, když je *sudý* a *souvislý*.

Pro významné typy grafů se používá vlastní název a označení. Název většinou plyne ze symetrie nebo vlastností grafu.

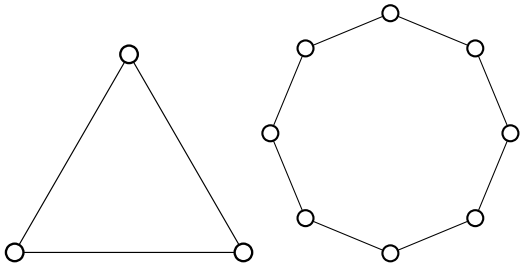
Definice 3.16 *Graf, jehož každé dva vrcholy jsou navzájem sousední, nazýváme* Kompletní graf *a značíme K_n, kde n je počet vrcholů kompletního grafu.*



Obrázek 5: Kompletní grafy K_4 a K_9

Kompletní graf, nazývaný také úplný graf, má úplnou množinu hran $E(K_n) = \binom{V(K_n)}{2}$. Počet hran v kompletním grafu na n vrcholech je tedy $\frac{n\,(n-1)}{2}$. Doplněk kompletního grafu je množina nezávislých vrcholů, jelikož podle definice 3.8 je $E(\overline{K_n}) = \emptyset$. Každý kompletní graf má všechny vrcholy nejvyššího možného stupně, je $(n-1)$-pravidelný.

Definice 3.17 *Graf s množinou vrcholů $V = \{x_1, x_2, \ldots, x_n\}$, kde $n \geq 3$, a množinou hran $E = \{x_1x_2, x_2x_3, \ldots, x_{n-1}x_n, x_nx_1\}$, se nazývá* cyklus *a značíme jej C_n.*



Obrázek 6: Cykly C_3 a C_8

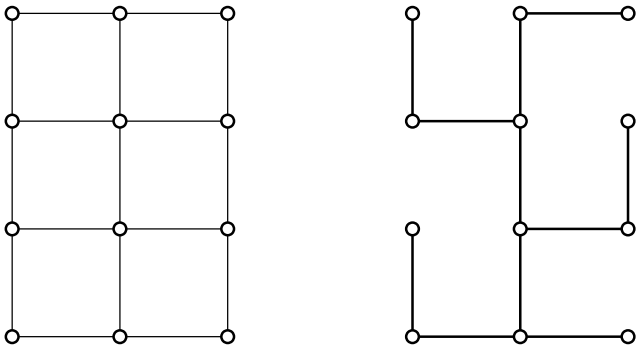
Cyklus je 2-pravidelný graf se stejným počtem vrcholů a hran $|E(C_n)| = n$. O počtu vrcholů n grafu C_n se často hovoří jako o délce cyklu. V tomto smyslu jsou na obrázku 6 cykly délky 3 a 8.

Cyklem v grafu G budeme rozumět takový podgraf grafu G, který je současně cyklem podle definice 3.17. Významnou roli mezi těmito cykly má takový cyklus který projde všemi vrcholy nadgrafu G.

Definice 3.18 *[1] Cyklus, který prochází všemi vrcholy grafu G, se nazývá* hamiltonovský cyklus *v grafu G. Graf, který obsahuje hamiltonovský cyklus, se nazývá* hamiltonovský graf.

Definice 3.19 *Graf se nazývá acyklický, jestliže žádný jeho podgraf není cyklus. Souvislý acyklický graf se nazývá* strom.

Definice 3.20 *Takový faktor grafu, který je současně stromem, se nazývá* kostrou *grafu.*



Obrázek 7: Graf G a jeho kostra K.

Nyní zavedeme pojmy a definice související s ohodnocením grafu.

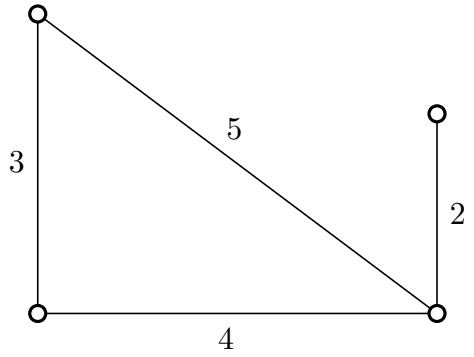
Definice 3.21 *[11] Ohodnocení grafu je funkce f, která vrcholům nebo hranám v grafu G přiřadí čísla, nejčastěji z množiny přirozených čísel.*

Přiřazená čísla mají většinou nějaký fyzikální nebo praktický význam, např. délky, kapacity, ceny.

Definice 3.22 *Hodnota vrcholu x nebo hrany xy grafu G je číslo přiřazené v ohodnocení f vrcholu x nebo hraně xy.*

Definice 3.23 *[11] Jsou-li v ohodnocení f přiřazeny hodnoty pouze hranám grafu $G = (V, E)$, hovoříme o* hranovém *ohodnocení grafu, v případě kdy jsou hodnoty přiřazeny pouze vrcholům, hovoříme o* vrcholovém *ohodnocení grafu.*

V této práci budeme používat pouze hranová ohodnocení grafu, jelikož hodnoty budeme přiřazovat pouze hranám v závislosti na vzdálenosti jejich koncových vrcholů.



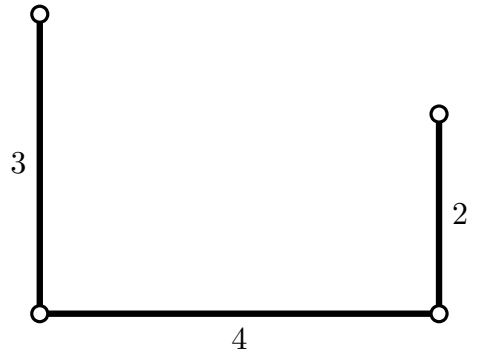
Obrázek 8: Příklad ohodnocení hran grafu H.

Definice 3.24 *Minimální kostra grafu G při ohodnocení f je taková kostra, že součet hodnot hran kostry je minimální možný.*

Definice 3.25 *[1] Nezávislá množina hran M grafu G se nazývá* párováním. *Říkáme, že koncové vrcholy hran v M jsou spárovány nebo M-saturované.*

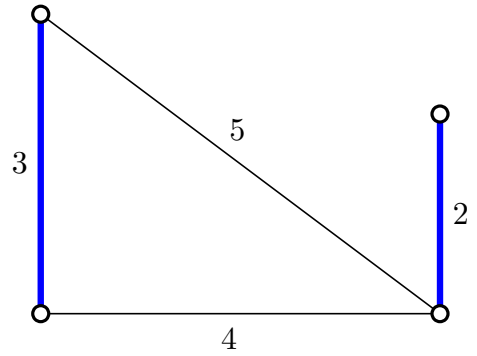
Párování budeme obvykle označovat písmenem M z anglického *matching*.

Definice 3.26 *[1] Párování M se nazývá* úplné, *jestliže je každý vrchol grafu G M-saturovaný.*



Obrázek 9: Minimální kostra grafu H z obrázku 8.

Definice 3.27 *Minimální úplné párování M grafu G je takové úplné párování, že součet hodnot přiřazeným hranám v párování M je ze všech možných úplných párování grafu G minimální.*



Obrázek 10: Minimální úplné párování grafu H z obrázku 8.

4 Problém obchodního cestujícího

It's addictive. No matter how much progress you make, you always have the nagging feeling that you still did not nail down a couple of hunches that could bring about another quantum leap.

– Vašek Chvátal, 1998.[2]

Ve třicátých letech dvacátého století [3] byl formulován tzv. Problém obchodního cestujícího, který je založen na úvaze, kdy obchodní cestující vyráží z určitého města, a během své cesty má za úkol navštivit několik dalších měst (většinou se požaduje aby každé město navštívil právě jednou) a potom se opět vrátit zpět. Řešením problému je nalezení takové cesty mezi městy, které má obchodní cestující navštívit, že jakákoliv jiná obchodní cesta bude stejně dlouhá nebo delší.

Představme si, že jste obchodní cestující z Ostravy. A máte navštívit všechna města v České Republice s počtem obyvatel nad 250 000. Máte tedy navštívit Prahu, Brno a Ostravu, kde začínáme. Jak to udělat aby výsledná cesta byla co nejkratší? No jednoduše pojedete z Ostravy do Prahy a potom do Brna a zpět do Ostravy. Nebo v obráceném pořadí Ostrava, Brno, Praha, Ostrava, kde délka cesty bude stejná jako v předchozím případě. Zde se zatím v tichosti předpokládá, že délka cesty z Ostravy do Prahy je stejná jako z Prahy do Ostravy. Jednoduché, další možnosti neexistují. Problém je tedy v tomto případě vyřešen a vy jako obchodní cestující, můžete vyrazit do Prahy nebo do Brna podle toho kam se více těšíte.

Poté co se vrátíte ze své první obchodní cesty, rozhodnete se navštívit při své další cestě všechna města v České Republice s počtem obyvatel nad 100 000. Takže nám přibude Plzeň, Liberec a Olomouc. A náhle je problém o dosti těžší, minule když jsme vyráželi z Ostravy jsme měli k dispozici dvě města a ať jsme si vybrali kterékoliv z nich následovala potom cesta do toho zbývajícího a zpět do Ostravy. Nyní si musíme vybrat z pěti měst a až do nějakého odcestujeme, tak se výběr možností sníží na čtyři, potom na tři atd. Celkový počet možných cest je nyní $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$, když přihlédneme ke skutečnost,i že pořadí navštívaných měst můžeme obrátit, aniž by se změnila délka cesty, dostáváme $120/2 = 60$. To sice ještě není nepřekonatelný problém ale porovnat 60 různých cest už zabere nějaký čas. Po návratu z druhé cesty se rozhodnete navštívit všechna města s počtem obyvatel nad 10 000, v roce 2011 jich v České republice bylo 132...

V roce 1962 firma Procter & Gamble vyhlásila soutěž o $ 10,000 , za což se dal tehdy postavit dům [3]. Zadání soutěže bylo následující:

[2] Citát z knížky: In Pursuit of the Traveling Salesman, [3].

[3] Dá se předpokládat, že se varianty tohoto problému řešili už dříve, i když třeba jen intuitivně.

Představte si, že Toody a Muldoon chtějí projet Spojené státy a navštívit při tom 33 lokací uvedených na mapě (viz Obrázek 11). Chtějí to ale udělat tak, aby celkem ujeli co nejkratší vzdálenost. Máte jejich cestu naplánovat z místa na místo tak, aby výsledná cesta byla co nejkratší. Začínáte v Chigagu, Illinois a také tam musíte skončit.

Obrázek 11: Soutěž firmy Procter & Gamble z roku 1962 [3]

Problém obchodního cestujícího, dále jen TSP (Traveling Salesman Problem), je klasická optimalizační úloha, kdy hledáme hamiltonovský cyklus s nejnižším součtem hodnot hran v kompletním hranově ohodnoceném grafu s n vrcholy. Budeme se zde zabývat pouze tzv. symetrickou variantou úlohy, kdy „cena“ hod-

nota hrany z vrcholu A do vrcholu B je stejná jako „cena“ hodnota hrany z vrcholu B do vrcholu A, ceny zde tedy mají charakter vzdáleností.

4.1 Problém obchodního cestujícího v metrickém prostoru

Variantou tohoto problému je problém obchodního cestujícího v metrickém prostoru, ve kterém vzdálenosti na grafu splňují trojúhelníkovou nerovnost, což je jeden z axiomů metrického prostoru. Toto zjednodušení odpovídá velkému množství reálných problémů (např. hledání na mapě), a zároveň umožňuje konstrukci aproximačních algoritmů. Tato varianta bude pro nás jistě zajímavá, jelikož systém na sebe kolmých cest ve skladu splnuje všechny axiomy metrického prostoru.

Definice 4.1 (metrického prostoru) *Metrický prostor je dvojice $(\mathcal{M}, \rho)$, kde $\mathcal{M}$ je libovolná neprázdná množina (v našem případě to bude množina vrcholů) a ρ je tzv. metrika, což je zobrazení*

$$\rho : \mathcal{M} \times \mathcal{M} \to \mathbb{R}$$

splňující následující axiomy:

1. *nezápornost* $\quad \rho(x, y) \geq 0,\ \rho(x, y) = 0 \Leftrightarrow (x = y) \qquad \forall x, y \in \mathcal{M}$
2. *symetrie* $\quad \rho(x, y) = \rho(y, x) \qquad \forall x, y \in \mathcal{M}$
3. *trojúhelníková nerovnost* $\quad \rho(x, z) \leq \rho(x, y) + \rho(y, z) \qquad \forall x, y, z \in \mathcal{M}$

Poznámka 4.2 V našem případě bude metrika ρ, zobrazení přiřazující každé dvojici vrcholů (x, y) hodnotu rovnou délce nejkratší (x, y)–cestě v grafu G.

Poznamenejme, že aproximační algoritmy předpokládají, že je možné se z libovolného vrcholu dostat do jakéhokoliv jiného vrcholu.

4.1.1 2–aproximační algoritmus

Problém obchodního cestujícího v metrickém prostoru lze přibližně řešit jednoduchým algoritmem[10] v polynomiálním čase. Algoritmus nejprve zkonstruuje minimální kostru grafu, např. podle *Jarníkova* nebo *Kruskalova* algoritmu, viz podkapitola 5.1. Z definice kostry plyne, že součet délky hran minimální kostry je menší (nebo roven) než součet délky hran v optimálním řešení TSP, jelikož minimální kostra obsahuje v nejhorším případě optimální řešení TSP bez nejdelší hrany (z důvodu acykličnosti).

V druhém kroku projde algoritmus kostru z libovolného vrcholu do hloubky a poznamená si všechny průchody přes vrcholy, protože se jedná o průchod do hloubky, budou zde některé vrcholy zpracovány vícekrát.

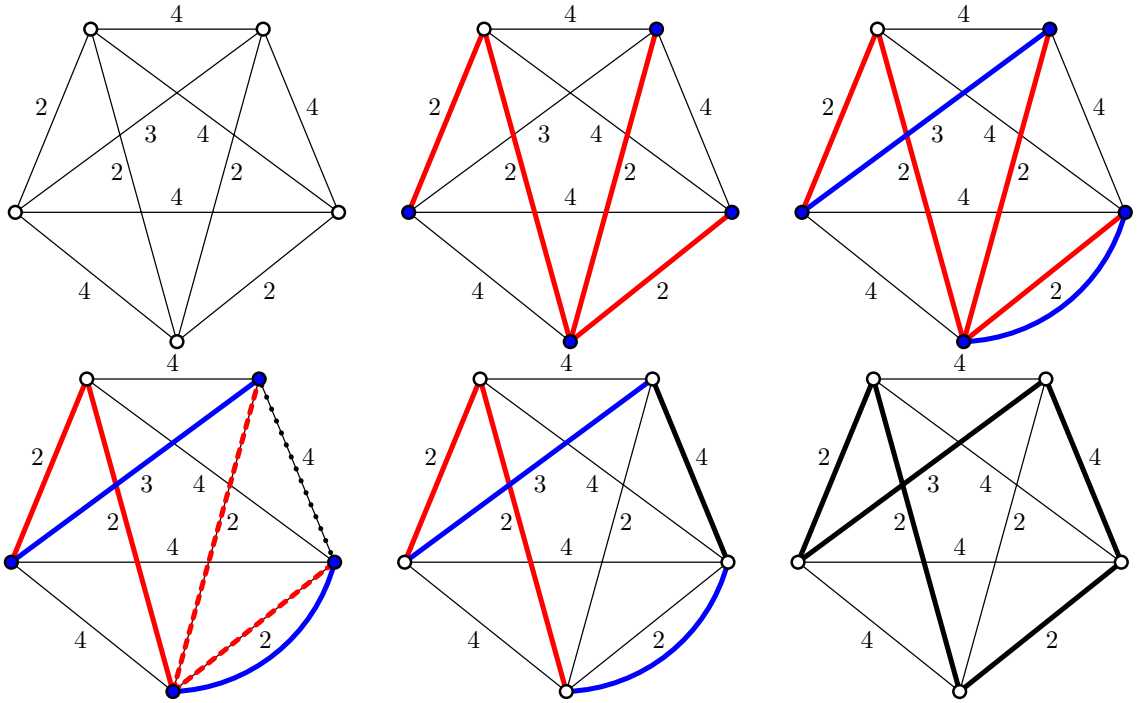
V posledním kroku algoritmus tento seznam projde a vynechá všechny duplicity a zanechá pouze první výskyty vrcholů. Tímto dojde k vytvoření samotného hamiltonovského cyklu. Protože v grafu platí trojúhelníková nerovnost, tak se cena výsledného hamiltonovského cyklu oproti původní minimální kostře maximálně zdvojnásobí a oproti optimálnímu řešení může být tudíž maximálně dvakrát delší.

4.1.2 Christofidův algoritmus.

Christofidův algoritmus řeší problém obchodního cestujícího v metrickém prostoru tak, že je výsledná trasa v nejhorším případě 1.5-krát delší než délka trasy optimálního řešení. Toto zlepšení je ovšem vykoupeno výrazně obtížnější implementací, a zároveň se na reálných datech ukazuje, že výsledek není v průměrném případě o mnoho lepší než při použití 2-aproximačního algoritmu uvedeného výše [10].

Christofidesův algoritmus nejprve zkonstruuje minimální kostru grafu T, např. podle *Jarníkova* nebo *Kruskalova* algoritmu. Z definice kostry plyne, že součet délky hran minimální kostry je menší (nebo roven) než součet délky hran v optimálním řešení TSP, jelikož minimální kostra obsahuje v nejhorším případě optimální řešení TSP bez nejdelší hrany (z důvodu acykličnosti kostry grafu). Poté kostru projde z libovolného vrcholu do hloubky a vybere ty vrcholy, jež mají lichý stupeň (vrcholů lichého stupně bude sudý počet, viz Věta 3.1) a zkonstruuje na nich kompletní graf K. V grafu K nalezne minimální úplné párování M, takové párování musí existovat, protože počet vrcholů je sudý a jedná se o kompletní graf. Představme si nyní, že budeme chtít vyřešit TSP na grafu K, řešení bude kratší nebo stejně dlouhé jako řešení na původním grafu. Vynecháním každé sudé resp. liché hrany z TSP řešení dostaneme dvě různá úplná párování, z nichž alespoň jedno z nich bude mít součet hodnot hran menší nebo rovno polovině délky optimálního řešení TSP. Součet délek všech hran v párování M bude tedy maximálně poloviční oproti optimálnímu řešení TSP. Hrany z minimálního úplného párování přidáme do minimální kostry.Vzniklý multigraf W, $V(W) = V(T)$ a $E(W) = E(T) \cup M$ je nyní eulerovský (tj. existuje v něm tah, který obsahuje všechny hrany multigrafu) a součet hodnot hran je maximálně 1.5 násobek řešení optimálního; 1-násobek za hodnoty hran minimální kostry a 0.5-násobek za hodnoty hran minimálního úplného párování.

Christofidův algoritmus v posledním kroku tento eulerovský tah projde a narazí-li na nějaký vrchol podruhé, přeskočí na nejbližší nenavštívený. Kvůli trojuhelníkové nerovnosti nemůže tímto přeskakováním zvýšit konečnou délku řešení, ta se tímto přeskakováním může případně snížit, nikoliv však pod teoretické minimum. Tímto dojde k vytvoření samotného hamiltonovského cyklu.



Obrázek 12: Ilustrace Christofidova algoritmu na grafu K_5.

4.2 Heuristické algoritmy

Zde by bylo dobré zmínit, že poměrně rozsáhlá třída algoritmů, které řeší problém obchodního cestujícího jsou tzv. heuristické algoritmy. Heuristické algoritmy většinou také připouští určitou míru nepřesnosti, na úkor toho jsou však schopny řešit i velice rozsáhlé problémy s počtem vrcholů několikanásobně převyšující náš problém. Heuristické algoritmy nemusí pro dva stejné vstupy dávat dva stejné výstupy a jejich implementace často zasahuje mimo teorii grafů. Z těchto důvodů jsme se rozhodli heuristické algoritmy jako jsou genetické algoritmy nebo mravenčí kolonie do této práce nezahrnout.

5 Teoretický rozbor

V minulé kapitole se objevil problém nalezení minimální kostry v grafu u aproximačních algoritmů řešící problém obchodního cestujícího. Ukážeme si tedy postupy jak minimální kostru grafu najít a zmíníme se také o Eulerovském grafu a minimálním párování, které budeme v budoucnu potřebovat. Poté se důkladně podíváme na graf, na kterém budeme zadaný problém řešit.

5.1 Algoritmy pro nalezení minální kostry grafu

Definice minimální kostry grafu byla zavedena v kapitole 3 na straně 13. Postup nalezení minimální kostry je často součástí řešení různých technických i jiných reálných problémů (například elektrifikace). Nalezení minální kostry řeší několik algoritmů, předpokládejme souvislý jednoduchý graf G a nezáporné ohodnocení hran f.

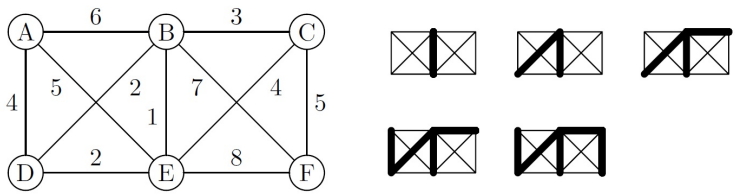
5.1.1 Kruskalův algoritmus

Algoritmus byl poprvé publikován Josephem B. Kruskalem, zaměstnancem Bellových Laboratoří, v roce 1956 [4]. Kruskalův algoritmus je příkladem hladového algoritmu. Hrany se prochází v pořadí od nejnižší hodnoty po nejvyšší a v případě, kdy přidáním hrany do řešení nevznikne v grafu cyklus, se hrana skutečně do řešení přidá.

Kruskalův hladový algoritmus hledá minimální kostru T na souvislém grafu G s hranovým ohodnocením f:

1. Řešení začíná jako faktor grafu G s prázdnou množinou hran, $E(T) = \emptyset$.
2. Seřaď hrany pohle jejih hodnoty, $f(e_1) \leq f(e_2) \leq \ldots \leq f(e_h)$.
3. Procházej hrany v pořadí podle velikosti. Pokud hrana e_i nevytvoří v grafu T cyklus, přidej hranu e_i do množiny $E(T)$.
4. Po projití všech hran je řešení T minimální kostrou grafu G.

Ve své práci z roku 1956 Kruskal popisuje variantu k uvedenému postupu, ve kterém se z grafu opakovaně odebírá hrana s největším ohodnocením, která ještě nezpůsobí, že se graf stane nesouvislým. Tímto ekvivalentním postupem můžeme také vytvořit minimální kostru grafu.



Obrázek 13: Příklad postupu nalezení minimální kostry pomocí Kruskalova algoritmu [5].

5.1.2 Jarníkův algoritmus

Jarníkův algoritmus (v zahraničí známý jako Primův algoritmus) pro hledání minální kostry poprvé popsal Vojtěch Jarník roku 1930, později byl nezávisle znovuobjeven roku 1957 Robertem Primem.

Algoritmus začíná s množinou (komponetou souvislosti) obsahující jediný (startovní) vrchol. Postupně do komponenty souvislosti přidává další vrcholy, které v ní ještě nejsou zahrnuty tak, že z přípustných možností vždy zvolí tu, jenž spojí nový vrchol s komponentou souvislosti hranou s nejnižším možným ohodnocením. Algoritmus skončí v momentě, kdy komponenta souvislosti obsahuje všechny vrcholy.

Jarníkův algoritmus hledá minimální kostru T na souvislém grafu G s hranovým ohodnocením f:

1. Nastavíme řešení T do počátečního stavu, $V(T_0) = r$, $E(T_0) = \emptyset$, kde r je libovolně zvolený startovní vrchol.

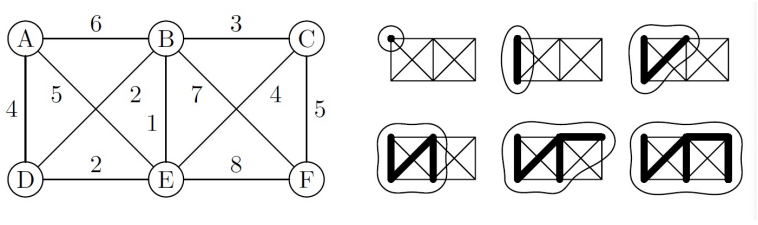
2. Z množiny hran $W = \{pq \in E(G) \,|\, p \in V(T_{i-1}), q \in V(G) \backslash V(T_{i-1})\}$ vyber tu s nejnižším ohodnocením $f(\{p, q\})$ a přidej ji do řešení, $E(T_i) = E(T_{i-1}) \cup \{p, q\}$, $V(T_i) = V(T_{i-1}) \cup \{q\}$.

3. Po $(n - 1)$ krocích máme řešení $T = T_n$, minimální kostru grafu G.

5.2 Minimální úplné párování

Definici minimálního úplného párování jsme zavedli v kapitole 3; definice 3.27. Minimální úplné párování budeme potřebovat v jednom kroku algoritmu hledající optimální trasu ve skladu.

Problém nalezení minimálního úplného párování nepatří mezi snadné úkoly. A ačkoliv je pro komplední grafy do řádu 10, a to bude prakticky bez vyjímky



Obrázek 14: Příklad postupu nalezení minimální kostry pomocí Jarníkova algoritmu [5].

náš případ, možné řešit tento problém hrubou silou, je dobré mít v záloze i sofistikovanější algoritmus. Jedním z takových, který by bylo možné použít, je Edmondsův algoritmus, který zde nebudeme popisovat. Popis algoritmu je možné najít v článku [9] nebo lépe v článku Edmonds' Minimum Weight Perfect Matching Algorithm [8].

Hrubou silou se v tomto případě myslí porovnat součty hodnot hran všech možností úplného párování, kterých v našem případě, kdy hledáme minimální úplné párování na kompletním grafu sudého řádu[4], bude $\prod_{i=1}^{n/2}(2i-1)$, kde n je počet vrcholů minimální kostry, které jsou lichého stupně. Pro hodnotu $n = 10$ dostáváme 945 možností, kdy pro každou možnost je potřeba sečíst hodnoty 5 hran, což průměrný dnešní počítač zvládne za zlomek sekundy. Pro práci se sowtwarem MATLAB© je možné využít již implementované funkce pro nalezení minimálního úplného párování, obsažené v toolboxu grTheory [7].

5.3 Eulerovský tah

Definici eulerovského tahu jsme zavedli v kapitole 3; definice 3.15. Eulerovský tah se nám bude v jedné fázi algoritmu hledající optimální trasu hodit, proto zde zmíníme postup jak eulerovský tah najít. Eulerovské grafy jsou grafy, o kterých můžeme říct, že je lze nakreslit "jedním tahem", tím myslíme, že tužku v průběhu kreslení nezvedneme z papíru, což přesně odpovídá definici 3.15 uzavřeného eulerovského sledu. Cesta vysokozdvižného vozíku po skladu je v určitém smyslu taky "jedním tahem", nebudeme ale hledat eulerovský tah na grafu reprezentující celý sklad, ale jen na určitém podgrafu, obsahující vrcholy reprezentující biny, které musíme navštívit.

Algoritmus vychází z Věty 3.2 na straně 11. Ověříme-li, že je graf (multigraf) sudý a souvislý, víme že bude obsahovat eulerovský tah, zbývá jen najít pořadí hran.

[4]Graf bude sudého řádu jelikož hledáme minimální párování jen mezi vrcholy minimální kostry, které jsou lichého stupně, a těch je podle Věty 3.1 sudý počet.

Algoritmus pro hledání uzavřeného eulerovského tahu na zadaném grafu, multigrafu G:

1. Ověříme, že je graf G souvislý, a že je sudý. Pokud tomu tak není, končí algoritmus výpisem, že graf uzavřený eulerovský tah neobsahuje.

2. Vybereme náhodně počáteční vrchol $u \in V(G)$.

3. Postupně konstruujeme $(u,u)-$tah tak, že se přesunujeme z aktuálního vrcholu do libovolného sousedního a hranu, kterou jsme k přesunu použili společně s koncovým vrcholem uložíme do tahu. Postupujeme tak dlouho než se znovu dostaneme na vrchol u. Tím dostaneme uzavřený $(u,u)-$tah.

4. Uzavřený $(u,u)-$tah zkontrolujeme. Postupně procházíme vrcholy v $(u,u)-$tahu, a u každého vrcholu x zkontrolujeme zda je incidentní s hranou, která nepatří do $(u,u)-$tahu. Jestliže ano, pak přerušíme kontrolu, $(u,u)-$tah ve vrcholu x rozpojíme a místo vrcholu x vložíme $(x,x)-$tah, který zkonstruujeme stejně jako jsme konstruovali $(u,u)-$tah v předchozím kroku. Sousední vrcholy ale vybíráme tak, aby jsme do $(x,x)-$tahu ukládali jen hrany, které nejsou součástí $(u,u)-$tahu, toto je díky splnění podmínek z prvního kroku vždy možné. Po vložení $(x,x)-$tahu pokračujeme v kontrole $(u,u)-$tahu.

5. Předposlední krok 4 opakujeme do doby, než kontrola proběhne bez nalezení vrcholu, který by byl incidentní s hranou, která nepatří do $(u,u)-$tahu.

5.4 Teoretický pohled na zadaný problém

Nyní vybaveni teoretickými znalostmi a známými algoritmy, se můžeme podívat na zadaný problém v novém světle a pokusit se ho zformulovat ve tvaru, se kterým se nám bude snadno manipulovat. V následujících odstavcích budu používat výraz "ulička", tímto budu dále myslet prostor mezi regály, kde se pohybuje vysokozdvižný vozík. Na mapě skladu (obrázek **??**) zobrazené na straně **??** v Neveřejné části to odpovídá vodorovným mezerám mezi žlutými regály. Pod pojmem "ulička"budou spadat i regály, které ji obklopují, budeme například hovořit o binech nebo materiálu, které jsou ve stejné "uličce". Cesty, které jednotlivé uličky spojují, budu nazývat spojovací cesty. Dále nebudeme brát v potaz patra skladu, jelikož je z pohledu nejkratší trasy naprosto lhostejné ve kterém patře se materiál k vyzvednutí nachází, vysokozdvižný vozík musí zastavit na stejném místě při vyzvednutí materiálu z třetího i pátého patra. Všechny patra tedy sloučíme do jednoho "přízemí", kde budeme problém řešit. Pokud bude při výsledném sestavování pořadí binů nutné nabrat různé materiály uložené ve

skladu v různých patrech nad sebou, naberou se tyto materiály těsně po sobě v pořadí od shora dolů, nebo případně v takovém, které nejvíce vyhovuje obsluze vysokozdvižného vozíku. Zde není možné nijak systematicky ušetřit čas.

Problém, tak jak jsme ho uvedli v kapitole 2, je třeba převést do řeči teorie grafů, abychom měli jednoznačně danou strukturu, se kterou budeme pracovat. Budeme-li hledat graf, který by situaci skladu dobře interpretoval, první co nás pravděpodobně napadne, bude reprezentovat každý bin ve skladu vrcholem a jednotlivé uličky a spojovací cesty vedoucí od jednoho binu k druhému, hranami. Tím získáme graf, který celkem výstižně popisuje daný sklad, obrázek grafu je uveden v neveřejné kapitole 9 na straně **??**. Pokud na něm budeme chtít daný problém řešit, budeme postupovat tak, že si označíme vrcholy, které reprezentují biny s materiálem k vyzvednutí. Tyto označené vrcholy potom budou tvořit povinné zastávky obchodního cestujícího, který bude grafem cestovat. Nevýhoda tohoto přístupu spočívá v zmíněné složitosti problému obchodního cestujícího pro graf s velkým počtem vrcholů.

Pokud se ale v objednávce vyskytnou požadavky na materiál, který je uložen v binech ve stejné uličce blízko sebe, nebo se nachází naproti, můžeme takovéto biny sloučit do jednoho vrcholu s konstatováním, že se doupouštíme určitého zjednodušení. Sloučením více binů do jednoho vrcholu způsobíme, že všechen materiál z objednávky nacházející se v binech sloučených do jednoho vrcholu bude nabrán společně, v pořadí které budeme specifikovat dále. To sebou přináší jednu velkou výhodu a to, že v rámci hledání optimálního pořadí binů považuju všechny biny sloučené do jednoho vrcholu za jeden jediný, a navštívit jakýkoliv z nich znamená navštívit všechny. Vyvstává otázka, zda toto zjednodušení může vnést do řešení určitou chybu. Pokud jde o případ, kdy se dva biny nacházejí v jedné uličce naproti sobě, tak jejich sloučením se žádné chyby nedopouštíme, vysokozdvižný vozík musí v každém případě přijet na stejné místo, kde se posléze otočí doleva či doprava, což trvá stejnou dobu. Při zpětném sestavování pořadí binů tedy není nutné brát v potaz, ve kterém ze dvou regálů ohraničující uličku se materiál nachází. Pokud jde o případ, kdy se biny nacházejí v jedné uličce blízko sebe, ale už ne naproti sobě, je situace složitější a zde si již nemůžeme být jisti, že se nedopustíme drobné chyby. Vzhledem k tomu, že optimální řešení jistě nebude obsahovat cestu, kde se budeme vracet na již navštívená místa pro "zapomenutý"materiál, můžeme očekávat, že tato případná chyba bude zanedbatelná, a díky výše popsanému zjednodušení nám výrazně klesne počet vrcholů v grafu reprezentující sklad a tím i výpočetní složitost celého problému.

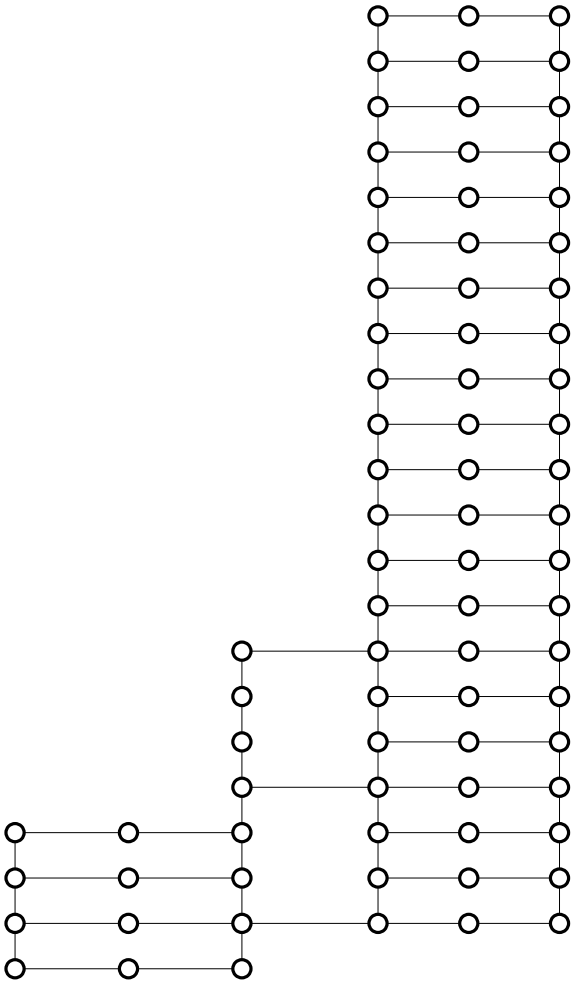
Tuto možnost sjednocení binů do jednoho vrcholu můžeme na mapě skladu provést na více místech s ohledem na rozmístění binů ve skladu. Vhodným příkladem jsou uličky, které mají pouze jeden vstup, který je třeba použít také i pro opuštění uličky, jelikož druhý konec tvoří stěna. V takovém případě můžeme všechny biny z této uličky reprezentovat jedním vrcholem, a v případě, kdy se

z jakéhokoliv z těhto binů nabere materiál, naberou se při této zastávce i všechy materiály z ostatních binů v této uličce, jsou-li nějaké obsaženy v objednávce. Je zřejmé, že do takovéto uličky není za žádných okolností výhodné vracet se podruhé. U uličky se dvěma vstupy/výstupy už tomu tak sice být nemusí, ale zahrneme-li do uváhy manévr, který musí obsluha vysokozdvižného vozíku provést, aby do uličky zatočila a následně z ní vyjela (při výjezdu z uličky je rovněž potřeba dávat pozor, aby nedošlo ke kolizi dvou vysokozdvižných vozíků), můžeme si situaci ulehčit tím, že i biny z takovéto uličky sjednotíme do jednoho vrcholu. Řešení nalezené v zjednodušeném grafu potom sice nemusí být nejlepší možné co se celkové vzdálenosti týče, předpokládáme však, že bude časově výhodnější nebo rovnocené. Zásadní výhodou tohoto přístupu ovšem je, že významně sníží počet vrcholů v grafu, kterým budeme sklad reprezentovat. Tím dojde ke snížení časové náročnosti algoritmu.

Nyní se podíváme na kostrukci grafu, který pro nás bude představovat model reprezentující sklad. Jelikož tento graf budeme kostruovat v závislosti na mapě skladu, bude tato část uvedena v neveřejné kapitole 9.

Výsledný graf, zobrazen na Obrázku 15, je vhodným zjednodušujícím modelem skladu. Strukturu skladu a pohyb mezi ”biny”ještě dobře popisuje podle skutečnosti, ale jen s využitím minimálního počtu vrcholů.

Při zpětném sestavování výsledného pořadí binů, využijeme postupu, kterým jsme sestavovali model skladu. Jelikož každý vrchol v grafu reprezentující sklad obsahuje vždy pouze biny z jedné uličky, která má jeden případně dva vstupy/výstupy, je vždy jedna z možností nabrat materiál zleva doprava (podle čísla pozice binu v uličce vzestupně) nebo zprava doleva (podle čísla pozice binu v uličce sestupně) optimální. Výběr jedné z možností je závisý na umístění předchozího vrcholu v optimalizovaném pořadí. Nacházel-li se předchozí vrchol na obrázku 15 vpravo od aktuálního, procházíme biny v sestupném pořadí, jinak procházíme biny ve vzestupném pořadí.



Obrázek 15: Zjednodušený graf popisující situaci ve skladu. Tento graf budeme používat jako model reprezentující sklad.

6 Optimalizace

Nyní si ukážeme algoritmus, pomocí kterého budeme cestu ve skladu optimalizovat. Zde by bylo vhodné zmínit, že pro deterministický algoritmus je zcela zásadní volba modelu reprezentující sklad, o čemž jsme pojednávali v podkapitole 5.4. Model musí být dostatečně zjednodušující a současně pokrýt všechny důležité aspekty pro hledání skutečné, fyzicky existující optimální trasy. Těmto kritériům bude zajisté vyhovovat více modelů. My jsme, po pečlivé uváze, zvolili model reprezentující sklad grafem na obrázku 15.

V této kapitole nejdříve popíšeme vlastní fungující program s konkrétními návrhy na řešení některých problémů. Poté si přiblížíme algoritmus v obecné podobě tak, aby mohl být implementovám v jakémkoliv programovacím jazyce. Při implementaci algoritmu jsme vycházeli z myšlenky Christofidova algoritmu, popsaném v podkapitole 4.1.2, a přizpůsobili jsme ho naší konkrétní situaci.

6.1 Implementace Algoritmu

Pro vlastní potřebu autora diplomové práce vznikal postupně program řešící optimalizaci cesty v konkrétním modelu skladu na reálných datech zaslaných firmou Mölnlycke Health Care. Ukázka vstupních dat je přiložena v kapitole 9 na straně **??**. Vstupní data osahovala několik objednávek, každá sestávající z několika komponent, kterým bylo přiřazeno číslo binu, kde se mají vyzvednout. Vstupní hodnoty tedy reprezentovaly body na mapě skladu. Očekávaný výstup potom měl obsahovat seřazení binů podél nejkratší cesty pro každou objednávku. Příklad výstupních dat je rovněž v kapitole 9.

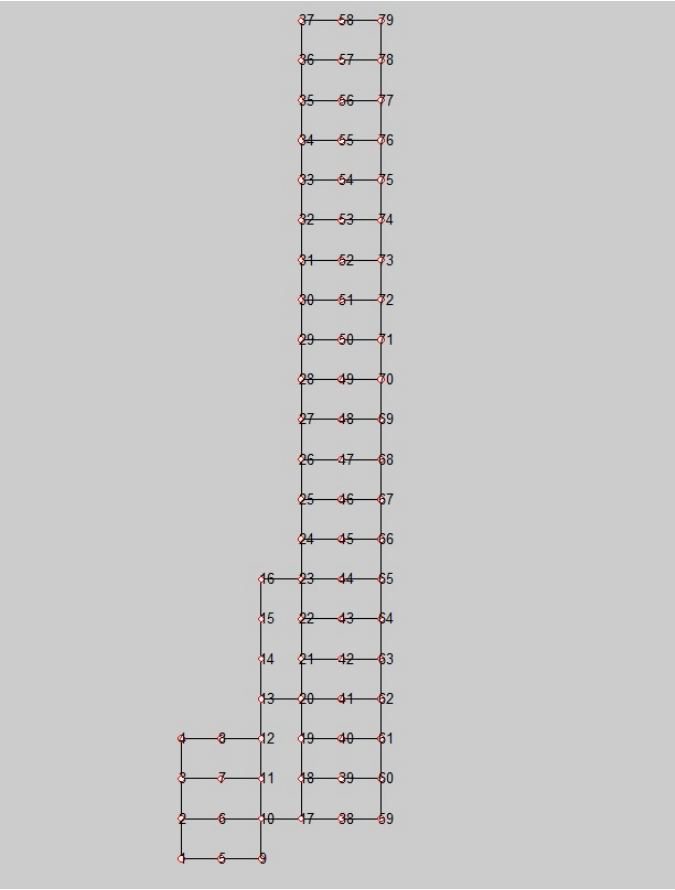
Program byl implementován v softwaru MATLAB©s použitím balíků Matgraph [6] a grTheory [7] pro práci s grafy a grafovými algoritmy. Tento software byl vybrán z důvodu, že již s ním byl autor práce dříve seznámen, algoritmus samotný je možné implementovat v libovolném programovacím jazyce.

Nejdříve bylo potřeba vytvořit graf G (obrázek 15 na straně 26), onen zmiňovaný model skladu, se kterým budeme dále pracovat,

```
g=graph;
%mrizka 22x6 (cesta krat cesta)
grid(g,22,6);
%odstranime hrany a vrcholy ktere do grafu nepatri
a=[67,89,111];
b=5:22;
c=27:44;
d=53:66;
r=[a,b,c,d]';
```

```
delete(g,r);
o1=38:57;
o2=39:58;
O=[o1',o2'];
delete(g,O);
p1=5:7;
p2=6:8;
P=[p1',p2'];
delete(g,P);
L=[11,18;12,19;14,21;15,22];
delete(g,L);
```

Výpis 1: vytvoří graf uvedený na obrázku 16.



Obrázek 16: Graf reprezentující sklad (vykreslen v MATLABu).

a sestavit matici D délek najkratší cesty,

$$D_{i,j} = \rho(i,j),$$

kde ρ je metrika zavedená v definici 4.1 resp. 4.2 na straně 17. Uložit si matici D je výhodné i přes to, že obsahuje $|(V(G))|^2$ prvků. V algoritmu budeme často potřebovat zjistit vzdálenost dvou vrcholů a v takovém případě bude stačit vybrat příslušný prvek z matice D.

```
D=dist(g);
```

Výpis 2: vytvoří matici D.

Načteme data zaslaná firmou Mölnlycke Health Care, resp. pouze tu část dat kterou budeme pro náš algoritmus potřebovat. Jedná se o tři sloupce ze souboru.xls obsahující číslo objednávky, pořadí položky v rámci objednávky a umístění položky ve skladu uvedením binu. Na obrázku **??** v neveřejné části – kapitola 9 jsou příslušné sloupce označeny barevně.

```
[L1,L2,L3]=nactip('VstupniData.txt');
```

Vstupní data se načítají z textového souboru, do kterého jseme si dříve uložili zmíněné tři sloupce dat. Každý ze sloupců si uložíme jako vektor čísel, resp. znaků.

```
function[objednavka,poradi,bin] = nactip( pickinglist )
% funkce nacte picking  list
 fid  = fopen( pickinglist );
mydata = textscan(fid,  '%n␣%n␣%s');
fclose( fid );
objednavka=mydata{1};
poradi=mydata{2};
bin=mydata{3};
```

Výpis 3: funkce pro načtení dat.

Nyní načtená data projdeme, oddělíme od sebe jednotlivé objednávky, a pro každou objednávku poté provedeme optimalizaci pořadí uvedených binů. My jsme použili následující jednoduchý postup u kterého jsme navíc kontrolovali zda se nejedná o poslední položku seznamu.

```
%seznam vrcholu grafu g, ktere musime projit v ramci jedne objednavky
u =[];
%seznam binu v ramci jedne objednavky
ubin={};
%pocet polozek v nactenem seznamu
 delkapickinglistu =length(L1);
%ukazatel v ramci objednavky
cislo_polozky=1;
for  ukazatel=1:delkapickinglistu
    if  ukazatel == delkapickinglistu
        %vypise se vse co zbyva
        %podobny postup jako v casti else s jednim volanim optimalizace_zapis navic.
```

```
    else
        if L2(ukazatel)==cislo_polozky
            %do objednavky se vlozi dalsi polozka
            polozka=L3(ukazatel);
            u(cislo_polozky)=bin(polozka{1});
            ubin(cislo_polozky)=L3(ukazatel);
            %cislo polozky se zvysi o 1
            cislo_polozky=cislo_polozky+1;
        else
            %ulozi cislo zkompletovane objednavky
            obj=L1(ukazatel-1);
            % zavola cast programu pro optimalizaci a zapis do souboru
            optimalizace_zapis

            % vzmaze stavajici hodnoty, priprava pro novou objednavku
            u=[];
            ubin={};
            ret=L3(ukazatel);
            % vlozi prvni polozku do nove objednavky
            u(1)=bin(ret{1});
            ubin(1)=L3(ukazatel);
            %nastavi ukazatelatel na druhou polozku
            cislo_polozky=2;
        end
    end
end
```

Výpis 4: procházení načtených dat.

Při procházení dat jsme pro každou objednávku vytvořili seznam vrcholů u, jehož prvky odpovídají vrcholům, které je při kompleteci objednávky třeba navštívit, jelikož jeden nebo více binů, které tento vrchol reprezentuje, obsahují položku z objednávky. Seznam vrcholů u obsahuje čísla, pomocí kterých je možno jednoznačně identifikovat vrchol z grafu G pomocí obrázku 16 (v algoritmu se k vrcholu přistupuje přímo přes číslo, pod kterým je ve struktuře grafu uložen).

Při načtení a následném rozdělení dat jsem měli k dispozici pouze seznam binů, tak jak nám ho poskytla firma Mölnlycke Health Care, viz obrázek **??** v neveřejné části na straně **??**. Ukážeme si, jak jsme v algoritmu vyřešili část, kterou jsme popsali v podkapitole 5.4, kdy pod jeden vrchol v námi zvolené modelové situaci spadá více binů. Pro tento účel je potřeba sestavit určitý překladač, který zadanému binu bude schopen přiřadit vrchol (číslo vrcholu), který tento bin v našem modelu reprezentuje. Problém zpětného překladu pak již není třeba řešit, neboť právě rozdělení binů do jednotlivých vrcholů bylo konstruováno tak, aby v rámci jednoho vrcholu bylo možné seřadit biny jen podle jejich uspořádání v uličce. Překladač v tuto chvíli překládá jen biny, které jsou zadány ve formátu *sklad ulička – patro – pozice v uličce*. V pickinglistu se vyskytují i biny v jiném

formátu (zbytkové zboží), ty ale bez přesné znalosti místa uložení, které v tuto chvíli nemáme k dispozici, nemohou být do optimalizace zařazeny.

```
function[n] = bin(s)
%funkce vraci cislo vrcholu reprezentujici zadany bin s
% vstup -- s retezec znaku
% vystup -- n cislo vrcholu
```

Výpis 5: funkce přiřadí binu vrchol grafu G. (funkční verze funkce je uvedena na konci neveřejné části práce)

V tuto chvíli máme připravený jak graf G, matici D obsahující vzdálenosti jednotlivých vrcholů grafu G tak i seznam vrcholů, které potřebujeme navštívit. Nejdříve si ukážeme část programu, která zapisuje výsledek do výstupního souboru, poté přejdeme k nejdůležitější části optimalizace, kterou tato část volá.

Následující část zdrojového kódu zapisuje optimálně seřazené biny z jednotlivých objednávek do souboru "*output2.txt*". Pro každou objednávku vytiskne její číslo a v případě, že je počet vrcholů k optimalizaci[5] alespoň 3 (1 nebo 2 vrcholy není třeba optimalizovat, jakékoliv "pořadí je optimální") zavolá funkci, která nalezne optimální pořadí vrcholů, podle kterého se potom vypíšou biny do výstupního souboru.

```
%Otevre soubor pro zapis
file_2 = fopen('output2.txt','a');
%Vytiskne cislo aktualni objednavky
fprintf(file_2,'----------OBJEDNAVKA %d----------\n',obj);
%Vytiskne biny z objednavky
for i=1:length(ubin)
    fprintf(file_2,'o     %s \n',ubin{i});
end
%Vytiskne vstupni vektor s vrcholy ktere se maji navstivit
for j=1:length(u)
    fprintf(file_2,' %d   ',u(j));
end
%Ulozi puvodni poradi vrcholu
uu=u;
vysledneporadi=[];
%Seradi vrcholy podle prirazenych cisel, odstrani duplicity a pripadne
%hodnoty 0.
u=unique(u);
if u(1)==0
    u=u(2:length(u));
end
%Overeni vstupnich podminek
%Pokud mame na vstupu 1 nebo 2 vrcholy, jsou uz optimalne serazene.
if length(u)<3
```

[5]Ve skutečnosti by stačilo optimalizovat pořadí 4 a více vrcholů.

```
        fprintf( file_2 , 'NIZKY-POCET-VRCHOLU \n');
        fprintf( file_2 , 'vysledne poradi vrcholu je:\n');
        fprintf( file_2 , ' -- %d',u);
        vysledneporadi=u;
    %Jinak se zavola funkce pro optimalizaci
    else
        %zavola funkci ktera najde optimalni poradi vrcholu
        %vstup funkce je graf g, matice vzdalenosti D a vektor vrcholu u.
        [poradi,delka,cas] = optimalizace(g,D,u);
        fprintf( file_2 , 'ooo------DIPLOMOVA-PRACE-------ooo\n');
        fprintf( file_2 , 'vysledne poradi vrcholu je:  \n');
        fprintf( file_2 , ' -- %d',poradi);
        fprintf( file_2 , 'delka je: %d \n',delka);
        fprintf( file_2 , 'cas nalezeni: %d \n',cas);
        vysledneporadi=poradi;
        % pokud je vrcholu mene nez 8 (jinak by jsme cekali prilis dlouho)
        % muzeme nasi funkci porovnat s funkci TSP pro nalezeni cesty
        % obchodniho cestujiciho, ktera je implementovana v toolboxu grTheory
        if length(u)<8;
            [TSPporadi,TSPdelka,TSPcas] = najdi_tsp(D,u);
            fprintf( file_2 , 'o---------TSP-MATLAB---------o\n');
            fprintf( file_2 , 'vysledne poradi TSP je:  \n');
            fprintf( file_2 , ' -- %d',TSPporadi);
            fprintf( file_2 , 'delka TSP je: %3.1f \n',TSPdelka);
            fprintf( file_2 , 'cas nalezeni TSP: %d \n',TSPcas);
            %a pokud by se vysledne delky tras odlisovaly, upozornime na to.
            if (int32(mdelka)) ~= (int32(TSPdelka))
                fprintf( file_2 , 'POZOR-DELKY JSOU RUZNE \n');
            end
        end
    end
    %Nyni zname poradi vrcholu a potrebujeme jeste vysledne poradi binu.
    count=1;
     listbin ={};
    %Seradime je podle poradi vrcholu a jeste v ramci ulicky,
    %posledni dve cisla binu udavaji jeho pozici v ulicce.
    for i=1:length(vysledneporadi)
        sprvni={};
        bprvni =[];
        pozicevrcholu=find(uu==vysledneporadi(i));
        for j=1:length(pozicevrcholu)
            sprvni{j}=ubin{pozicevrcholu(j)};
        end
        for jj =1:length(pozicevrcholu)
            a_bin=sprvni{jj };
            bprvni(k)=((10*(str2double(a_bin(7))))+(str2double(a_bin(8))));
        end
        [A,X]=sort(bprvni);
        for jjj =1:length(pozicevrcholu)
```

```
            listbin {count}=sprvni{X(jjj)};
            count=count+1;
        end
    end
    % vypiseme vysledne poradi binu.
    fprintf( file_2 , 'Vysledne poradi binu je: \n');
    for  i=1:length( listbin )
        fprintf( file_2 , 'x     %s \n', listbin {i});
    end
    fclose( file_2 );
```

Výpis 6: vytiskne požadovaný výsledek optimalizace do souboru

Uvedená část 6 zdrojového kódu v jedné fázi volá funkci `najdi_tsp`, která připraví vstupní parametry pro funkci TSP a tu poté zavolá. Funkce TSP, o které jsme zde již mluvili, je implementována v toolboxu grTheory a používali jsme ji k porovnání výsledků s našim algoritmem. Funkce `najdi_tsp` byla volána jen v případě, že byl počet vrcholů které je třeba projít, menší než 8, jinak by se na výsledek této funkce čekalo pro každou objednávku i několik minut.

```
function [pTS,fmin]=TravSalePro(C)
% Function [pTS,fmin]=grTravSale(C) solve the nonsymmetrical
% traveling  salesman problem.
% Input parameter:
%   C(n,n) − matrix of  distances between cities,
%      maybe, nonsymmetrical;
%      n − number of cities.
% Output parameters:
%   pTS(n) − the order of  cities ;
%   fmin − length of  way.
% Uses the reduction to integer  LP−problem:
% Look: Miller  C.E.,  Tucker A.  W., Zemlin R. A.
% Integer Programming Formulation of Traveling Salesman Problems.
% J.ACM, 1960, Vol.7, p. 326−329.
% Author: Sergii  Iglin
% e−mail: siglin@yandex.ru
% personal page: http:// iglin .exponenta.ru
```

Výpis 7: komentář k funkci TSP

Nyní se už konečně podíváme na samotnou funkci `optimalizace`, která zadaný problém řeší a hledá optimální pořadí vrcholů zadaných ve vektoru u. Funkce `optimalizace` požaduje jako vstupní hodnoty graf G, reprezentující model skladu, matici D, s délkou nejkretší (i, j)-cesty v grafu G uloženou v i-tém sloupci a j-tém řádku a seznam vrcholů které je třeba navštívit. Výstupní hodnoty jsou potom optimální pořadí vrcholů, celková délka trasy a doba běhu algoritmu. Funkce `optimalizace` používá funkce implementované v toolboxu grTheory

grMinSpanTree() funkce pro nalezení minimální kostry. Algoritmy pro nalezení minimální kostry jsme diskutovali v podkapitole 5.1

grMaxMatch() funkce pro nalezení maximálního párování. Minimální párování můžeme získat pomocí párování maximálního, pokud od pevně zvolené konstanty odečteme hodnotu hran. Problém minimálního párování jsme diskutovali v podkapitole 5.2

euler_trail() funkce pro nalezení eulerovského tahu. Eulerovský tah jsme probrali v podkapitole 5.1

```
function[u_poradi,delka,cas] = optimalizace(g,D,u)
%--------vystupy
%poradi=vysledne poradi vrcholu.
%delka= celkova delka trasy.
%cas=doba trvani algoritmu.
%--------vstupy
%g-graf g, sklad.
%D-matice vydalenosti.
%u-vektor obsahujici cisla vrcholu, ktere se maji navstivit .

%zacneme merit cas
tic;
l=length(u);
%vytvori matici E, ktera reprezentuje hrany kompletniho grafu na
%vrcholech z vektoru u.
%Kaydy radek matice E reprezentuje jednu hranu.
%V prvnich dvou sloupcich jsou ulozeny koncove vrcholy hrany ve tretim
%sloupci je potom delka teto hrany odpovidajici delce cesty ve skladu.
E=zeros((l*(l-1)/2),3);
c=1;
for i=1:(l-1)
    for j=(1+i):l
        E(c,1)=u(i);
        E(c,2)=u(j);
        E(c,3)=D(u(i),u(j));
        c=c+1;
    end
end
%Najde minimalni kostru k kompletniho grafu na vrcholech u, s ohodnocenymi
%hramami podle matice E.
%Volame metodu z toolboxu grTheory, ktera najde minimalni kostru grafu, viz
%Kruskaluv algoritmus v kapitole 5.1.1
t=grMinSpanTree(E);
k=graph;
copy(k,g);
label(k);
clear_edges(k)
```

```
for q=1:(length(t))
    fp=find_path(g,E(t(q),1),E(t(q),2));
    for z=1:(length(fp)-1)
        add(k,fp(z),fp(z+1));
    end
end
%ulozi delku souctu hran v minimalni kostre k.
mdel1=ne(k);
%score grafu (minimalni kostry) k.
stup=deg(k);
%najdeme vrcholy licheho stupne.
liche =[];
cc=1;
for i=1:(length(stup))
    if mod(stup(i),2)==1
        liche(cc)=str2double(get_label(k,i));
        cc=cc+1;
    end
end
%Nyni budeme hledat minimalni uplne parovani na mnozine vrcholu L minimalni
%kostry k, ktere jsou licheho stupne(tech bude sudy pocet).
%Matice F reprezentuje hrany kompletniho grafu na vrcholech L.
%V prvnich dvou sloupcich jsou ulozeny koncove vrcholy hrany ve tretim
%sloupci je potom delka teto hrany odpovidajici delce cesty ve skladu.
lll =length(liche);
F=zeros((lll*(lll-1)/2),3);
ccc=1;
for i=1:(lll-1)
    for j=(1+i):lll
        F(ccc,1)=liche(i);
        F(ccc,2)=liche(j);
        F(ccc,3)=D(liche(i),liche(j));
        ccc=ccc+1;
    end
end
%Jelikoz budeme volat funkci pro nalezeni maximalniho parovani, ale my
%potrebujeme parovani minimalni, odecteme stavajici hodnotu hran v matici F
%od konstanty, my zvolime konstantu 30, a nasledne ziskane hrany parovani
%budou tvorit minimalni parovani.
fv=(size(F));
for s=1:(fv(1))
    F(s,3)=30-F(s,3);
end
%Volame metodu z toolboxu grTheory, ktera najde minimalni parovani.
M=grMaxMatch(F);
%ulozime hrany minimalniho parovani a secteme jejich delku, nezapomeneme ze
%jsme predtim odecitali skutecnou hodnotu hran od konstanty 30.
m=graph;
copy(m,g);
```

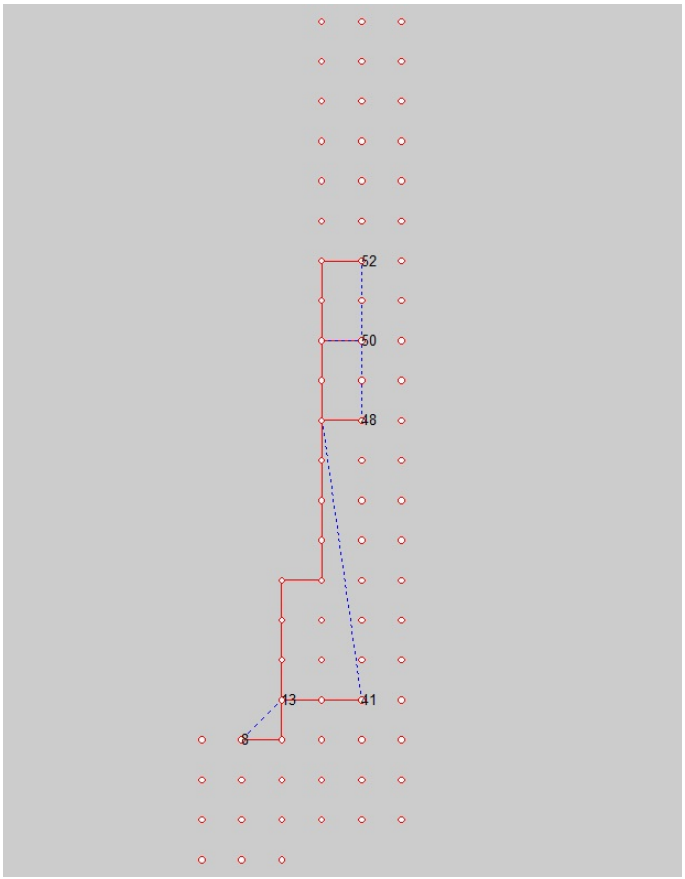
```
label(m);
clear_edges(m)
mdel2=0;
for p=1:(length(M))
    add(m,F(M(p),1),F(M(p),2));
    mdel2=mdel2+(30-(F(M(p),3)));
end
%Vytvorime multigraf spojenim minimalni kostry k a minimalniho parovani m.
eg=graph;
copy(eg,k)
union(eg,k,m);
%Vznikly multigraf je sudy, nalezneme Eulerovsky tah.
%Volame metodu z toolboxu grTheory, ktera najde Eulerovsky tah.
Eporadi= euler_trail (eg);
%ziskame poradi hran, pomoci koncovych vrcholu techto hran dostaneme poradi
%vrcholu v Eulerovskem tahu.
poradi=Eporadi(:,1);
%Nyni z poradi vrcholu odstranime ty vrcholy ktere nas nezajimaji,
%to jsou ty ktere nebyly obsazeny ve vstupu u.
u_poradi =[];
f=1;
for i=1:(length(poradi))
    for j=1:(length(u))
        if poradi(i)==ut(j)
            u_poradi(f)=mporadi(i);
            f=f+1;
            u(j)=0;
        end
    end
end
%secteme delku minimalni kostry a minimalniho parovani.
delka=mdel1+mdel2;
%ulozime jak dlouho algoritmus bezel.
cas=toc;
```

Výpis 8: funkce vrátí optimální pořadí vrcholů a celkovou délku trasy.

Výše popsaný program jsem otestoval na vstupních datech, která zaslala firma Mölnlycke Health Care. Vstupní data obsahovala 121 objednavek a na počítači s procesorem Intel Core i5 $3,2$GHz a 4GB RAM trval přibližně 9 sekund. Ve všech případech, kdy se výsledek porovnával s výsledkem funkce TSP z toolboxu grTheory, byly délky navrhovaných tras obou algoritmů stejné.

6.1.1 Možná vylepšení

V některých případech, nastane situace, kdy algoritmus najde takové řešení, které se lidskému vnímání nezdá příliš vhodné.

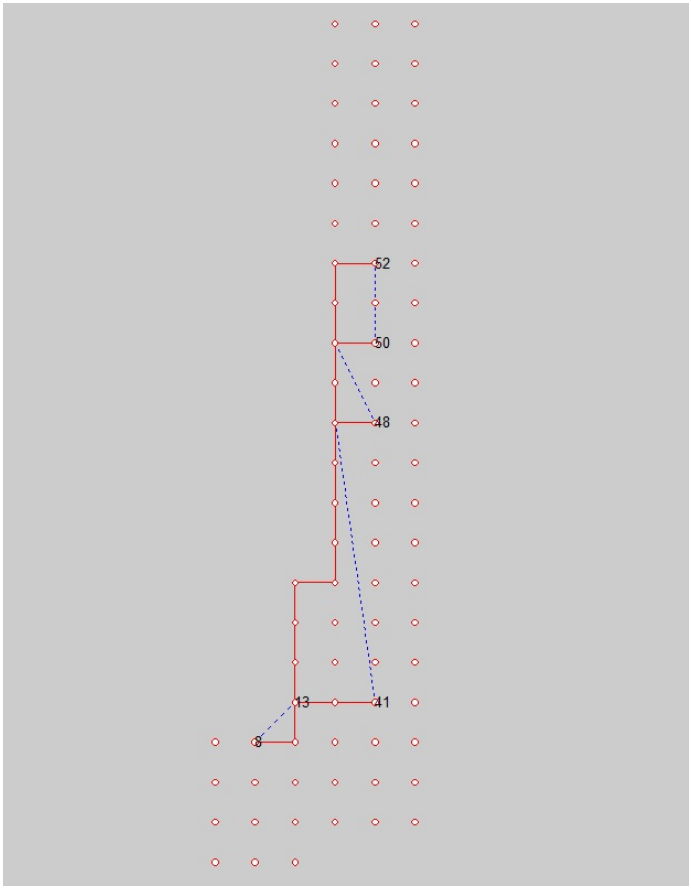


Obrázek 17: Příklad diskutabilního řešení s nalezeným pořadím $8, 13, 50, 52, 48, 41$.

A k tomuto řešení často existuje jiné, které je stejně dlouhé jako původní, ale pro člověka už není tolik "kontroverzní". Řešení nalezené na obrázku 17 s pořadím vrcholů $8, 13, 50, 52, 48, 41$ je sice v síti pravoúhlých uliček správně, je ale trochu nepříjemné, že z trojice vrcholů $48, 50, 52$ nejdříve jdeme do prostředního vrcholu 50 potom do vrcholu 52 a potom se "vracíme"do vrcholu 48. Mnohem přirozenější by bylo navštívit je popořadě $48, 50, 52$ nebo případně v pořadí obráceném. Na tomto místě však poznamenejme, že celková délka cesty bude ve všech těchto případech stejná. Tento drobný problém nastává ve chvíli, kdy algoritmus hledá minimální párování, které mnohdy není jednoznačne určeno. V této chvíli bychom chtěli donutit algoritmus aby upřednostnoval některá řešení

před jinými. Konkrétně bychom chtěli, aby se v minimálním párování nevyskytovaly příliš dlouhé hrany, které dovolují určité úseky, do kterých je potom nutné se "vracet", přeskočit. Toho můžeme dosáhnout tak, že k extrémně dlouhým hranám přičteme nějakou konstantu, která bude vyšší než konstanta přičtená k hranám kratším. Celková hodnota konstanty by ale měla být nižší než je nejratší možná délka hrany, tedy 1, abychom nemuseli hlídat, zda jde skutečně ještě o minimální párování. Zde se nabízí přičíst druhou mocnimu délky hrany (nejdelší hrana má délku 27.) vydělenou 1000. V takovém případě budou zachovány výše popsané požadavky a algoritmus do minimálního párování nevybere extrémně dlouhé hrany.

Po této úpravě nám dá program následující řešení, které už je "pocitově"přijatelné.



Obrázek 18: Nalezené řešení po úpravě $8, 13, 52, 50, 48, 41$.

6.2 Algoritmus

Nyní uvedeme v obecné podobě algoritmus, který najde optimální pořadí zadaných vrcholů u, které reprezentují bin, nebo více binů z objednávky. Algoritmus pracuje s modelem skladu (obrázek 15), který jsme odvodili v kapitole 5.4. Model skladu je stěžejní bod optimalizace, a kdyby byl navržen jinak, nemusel by dále popsaný algoritmus korektně fungovat.

Algoritmus

1. Algoritmus má dva vstupní parametry: graf G, představující model skladu a množinu vrcholu u, což jsou vrcholy, které v modelu reprezentují biny ze kterých je potřeba nabrat materiál pro objednávku, $u \in V(G)$.

2. Sestavíme matici D délek kejkratších cest mezi vrcholy $i, j \in V(G)$.
   $D_{i,j} =$ nejkratší (i, j)-cesta v grafu G.

3. Vygenerujeme kompletní graf K_u na u vrcholech a každou hranu tohoto grafu ohodnotíme $f(\{i, j\}) = D_{i,j}, \quad \forall\{i, j\} \in E(K_u)$ (přiřadíme hodnotu délky nejkratší (i, j)-cesty v grafu G).

4. V grafu K_u nalezneme minimální kostru za použití *Kruskalova* algoritmu, viz kapitola 5.1.

5. V grafu G nalezneme strom T, odpovídající minimální kostře grafu K_u. Hrany minimální kostry převedeme na jim odpovídající cesty v grafu G. Množina hran $E(T)$ bude obsahovat sjednocení hrany, které jsou součástí těhto cest (duplicitní hrany budou zahrnuty pouze jednou, nebude se jednat o multigraf).

6. Vrcholy lichého stupně grafu T označíme W, $W \subseteq V(G)$. Poznamenejme, že podle principu sudosti bude těhto vrcholů sudý počet $|W| \equiv 0 \pmod 2$.

7. Vygenerujeme kompletní graf K_W na W vrcholech a každou hranu tohoto grafu ohodnotíme $f(\{i, j\}) = D_{i,j}, \quad \forall\{i, j\} \in E(K_W)$. (přiřadíme hodnotu délky nejkratší (i, j)-cesty v grafu G)

8. Nalezneme minimální úplné párování M vrcholů W na grafu K_W.

9. Sjednocením hran $E(T)$ a M dostaneme sudý multigaf H, který bude souvislý.

10. Najdeme uzavřený Eulerovský tah $\mathcal{E}$.

11. Vrcholy u seřadíme podle jejich výskytu v Eulerovském tahu $\mathcal{E}$, pokud se vrchol v tahu nachází vícekrát[6], ponecháme pouze jeho první výskyt.

Tím získáme výsledné pořadí vrcholů. Výsledné pořadí binů potom získáme postupem, kdy procházíme vrcholy v nalezeném vysledném pořadí a u každého vrcholu vypíšeme všechny biny, které daný vrchol pro danou objednávku reprezentují. Rozdělení binů do vrcholů jsme při tom konstruovali úmyslně tak šikovně, že pokud je předchozí vrchol ohodnocen nižším číslem vypisujeme biny podle pozice v uličce vzestupně, pokud je předchozí vrchol ohodnocen vyšším číslem vypisujeme biny podle pozice v uličce sestupně. Ohodnocení vrcholů lze nalézt na obrázku 16 na straně 28. Toto má navíc význam jen pro vrcholy $5, 6, 7, 8, 10, 13, 16, 17, 20, 23$ a vrcholy 38 až 58 kde jsou biny umístěné v uličce do které je možno vjet z obou stran. U všech ostatních vrcholů toto nehraje roli, jelikož ve slepé uličce je vždy potřeba dojet až pro materiál, který je od vjezdu do uličky nejvzdálenější, a po cestě zpět (nebo po cestě tam) nabrat postupně materiál zbývající.

[6]Tato situace může teoreticky nastat, například mezi skladem B a C jsou pouze tři průchody a je možné, že nejvýhodnější bude použít jeden z nich dvakrát.

7 Závěr

Tato diplomová práce řeší problém optimalizace trasy pro vyzvednutí jednotlivých položek ve skladu, zadaný firmou Mölnlycke Health Care. Pro zpracování diplomové práce poskytl zadavatel informace, které považuje za důvěrné (např. mapa skladu) a nepřeje si jejich zveřejnění. Z tohoto důvodu je diplomová práce rozdělena na část veřejnou a neveřejnou. Diplomová práce je koncipována tak, aby případný čtenář byl schopen pochopit námi zvolený přístup k problému a jeho řešení i bez přístupu k neveřejné části práce.

V práci jsme zadaný problém pečlivě zformulovali a popsali jsme situaci ve skladu. Samostatnou kapitolu 3 jsme věnovali pojmům a definicím z teorie grafů, které jsme v průběhu celé práce používali a které byly potřeba při formulování teoretického přístupu k problému. Navrhli jsme určitá zobecnění a zjednodušili jsme teoretický model zadaného praktického problému tak, aby obsahoval pouze informace mající vliv na nalezení nejkratší trasy. Informace, které měli žádný, nebo zanedbatelný vliv na hledaný výsledek, nebyly do modelu zahrnuty. Systém, jakým byl model reprezentující sklad sestaven, má zásadní význam při optimalizaci a tvoří významnou část diplomové práce. Dále jsme si představili problém obchodního cestujícího a aproximační algoritmy, které ho řeší. Nakonec jsme popsali algoritmus pro řešení zadaného problému a s použitím částí zdrojového kódu aplikace, psané v jazyce MATLAB©, popsali postup jeho implementace.

Aplikace v jazyce MATLAB©, která pro zadanou objednávku najde optimální pořadí binů, není součástí této práce s ohledem na skutečnost, že firma Mölnlycke Health Care projevila zájem o nezveřejnění důvěrných informací, které jsou v některých částech zdrojového kódu obsaženy. Mimo to, vlastní aplikace nebyla zamýšlena jako výstup této diplomové práce, spíše sloužila k odzkoušení navrhovaného algoritmu a zjištění přibližné doby běhu algoritmu na reálných vstupních datech, což se podařilo.

Cíle práce bylo dosaženo s konstatováním, že zvolený algoritmus negarantuje sice nalezení naprosto nejkratší cesty, ale při testování na datech z reálného provozu, byly délky všech nalezených tras shodné s délkami tras, které našel TSP algoritmus implementovaný v toolboxu MATLABu [7]. Doba běhu obou algoritmů byla sice pro objednávky obsahující malý počet komponent řádově stejná, už však pro objednávky s počtem a umístěním jednotlivých komponent tak, že je nutné navštívit přibližně deset různých míst ve skladu, nachází námi navrhovaný algoritmus řešení v řádu sekund, zatímco zmiňovaný TSP algoritmus v řádu minut. Užití navrhovaného algoritmu samozřejmě není nijak vázáno na software MATLAB©a stejné výsledky můžeme očekávat v libovolném programovacím jazyce. Samotná implementace algoritmu je dle dohody v kompetenci firmy Mölnlycke Health Care, protože bude součástí jejich stávajícího softwaru.

Tato práce také ukazuje praktické využití teorie grafů při řešení reálného problému v průmyslu a možnost spolupráce akademické obce se soukromým sektorem. Na tuto spolupráci se dá v případě zájmu Mölnlycke Health Care navázat s návrhem lepší organizace uložení komponent ve skladu, aby se minimalizovaly délky tras při kompletaci objednávek.

8 Literatura

[1] P. KOVÁŘ, Teorie grafů (učební text), *VŠB-TU Ostrava*, 2012.

[2] D. L. APPLEGATE, R. E. BIXBY, V. CHVÁTAL, W. J. COOK, The Traveling Salesman Problem: A Computational Study, *Princeton University Press, 606pp*, 2006.

[3] W. J. COOK, In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation, *Princeton University Press*, 2012.

[4] J. B. KRUSKAL, JR., On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society*, 1956.

[5] J. ČERNÝ, Základní grafové algoritmy. *KAM, MFF UK - Praha*, 2010, *Verze 0.95.*

[6] E. SCHEINERMAN, Matgraph –*matlab toolbox for working with simple, undirected graphs.* 2008 *(použitá verze 2012)*. http://www.ams.jhu.edu/ ers/matgraph/

[7] S. IGLIN, grTheory –*Graph Theory Toolbox with functions for different tasks of graph theory for MATLAB©*, 2003 *(použitá verze 2012)*.

[8] L. GODDYN, Edmonds' Minimum Weight Perfect Matching Algorithm, *Math 408*, 2012. http://people.math.sfu.ca/ goddyn/Courseware/edmonds.pdf

[9] H. GABOW, An Efficient Implement at ion of Edmonds' Maximum – Matching Algorithm, *Stanford University*, 1972 *(Technical Report No. 31)*. ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/72/328/CS-TR-72-328.pdf

[10] Algoritmy.net [online], Problém obchodního cestujícího, 2012. Dostupné z: http://www.algoritmy.net/article/5407/Obchodni-cestujici

[11] P. KOVÁŘ, On Magic Graph Labelings, *Ph.D. Thesis*, *VŠB-TU Ostrava*, 2004.

[12] R. E. BURKARD et al. , Well-solvable special cases of the traveling salesman problem: a survey, *Society for Industrial and Applied Mathematics, Vol. 40, No. 3*, 1998.

9 Neveřejná část

Tato verze diplomové práce je veřejná a s ohledem na to v této verzi nejsou umístěné citlivé informace firmy Mölnlycke Health Care. Veškeré informace, na které se odkazujeme v textu výše, jsou uvedené v neveřejné verzi diplomové práce.